2/PRTS.

Translation of PCT Application as Filed

CONTROLLED EXECUTION OF A PROGRAM THAT IS INTENDED FOR A VIRTUAL MACHINE ON A PORTABLE DATA CARRIER

[0001] The invention relates generally to the technical field of executing a program that is intended for a virtual machine, on a portable data carrier that has a processor. A portable data carrier of that kind may be especially a chip card in various forms or a chip module. More specifically, the invention relates to the controlled execution of a program in order to detect faults or attacks and in order to prevent the security of the portable data carrier from being compromised by such faults or attacks.

[0002] Portable data carriers that have a virtual machine for executing programs are known, for example, under the trademark *Java Card*™. Such data carriers are described in Chapter 5.10.2 of the book "Handbuch der Chipkarten" by W. Rankl and W. Effing, Hanser Verlag, 3rd edition, 1999, pages 261 to 281. A detailed specification of the *Java Card* standard, the virtual machine JCVM (*Java Card Virtual Machine*) used therewith and of the programs (*Java Card Applets*) that are executed is to be found on the Internet pages of the company Sun Microsystems, Inc., at *http://java.sun.com/products/javacard.*

[0003] Portable data carriers are frequently used for applications where security is crucial, for example in connection with financial transactions or in electronic signature of documents. Techniques for attacking portable data carriers have already become known in which the execution of a program is disrupted by external interference. Such disruption may be caused, in particular, by voltage pulses, by the effect of heat or cold, by electric or magnetic fields, electromagnetic waves or particle radiation. For example, it is possible to alter register contents in the processor or memory contents by directing flashes of light onto the exposed semiconductor chip. Such interference may possibly

compromise the security of the data carrier if, for example, the data carrier outputs a defectively encrypted text which, when analysed, allows inferences to be made about a secret key.

[0004] There is therefore the problem of safeguarding a data carrier of the kind mentioned in the introduction from being compromised by attacks that interfere with the execution of a program by a virtual machine.

[0005] GB 2 353 113 A discloses a computer network that is capable of compensating for software faults to a certain extent. At least two computers, each executing a virtual machine, are provided in that computer network. If one of the virtual machines is found to be operating incorrectly, execution of the program is continued by the other virtual machine or machines.

[0006] The system known from GB 2 353 113 A is foreign to the generic type in question here, since it is intended not for a portable data carrier but for a complex network comprising a plurality of computers. The virtual machines are executed by a plurality of processors which are only loosely coupled to one another. Execution of the program is continued even when one virtual machine is disrupted. That teaching is not suitable for application in a portable data carrier having a single processor.

[0007] The object of the invention is accordingly to avoid the problems of the prior art at least to some extent, and to provide a technique for the controlled execution of a program, the program being intended for a virtual machine, on a portable data carrier, by means of which technique security risks in the event of an attack or a malfunction are avoided. In preferred embodiments, reliable protection is to be achieved with as little loss of performance of the portable data carrier as possible.

[0008] According to the invention, that object is completely or partially achieved by a method having the features of claim 1, by a portable data carrier having the features of claim 10 and by a computer program product having the features of claim 11. The dependent claims relate to preferred embodiments of the invention.

[0009] The invention is founded on the basic idea of having the one processor of the portable data carrier execute a plurality of virtual machines which in turn execute one and the same program. This measure provides a redundancy in the program execution, which can be utilised to detect malfunctions. It is a surprising result of the present invention that such a redundancy can also be obtained in a portable data carrier that has only a single processor.

[0010] According to the invention, the execution of the program is aborted if a difference is found between the operating states of the virtual machines. An attempt is not made, therefore, to identify one of the virtual machines as operating correctly and to continue execution of the program with that virtual machine. By aborting the program as provided according to the invention, especially high security against attacks is achieved.

[0011] The invention offers the considerable advantage that it can be implemented without any difficulty on conventional hardware. In addition, no adaptation of the program that is to be executed to the attack protection according to the invention is required. All programs intended for the standard virtual machine will run unchanged on the data carriers configured in accordance with the invention, which greatly furthers the acceptance of the invention.

[0012] When the operating states of the virtual machines are being checked for correspondence, the comparison that takes place is preferably not a complete

comparison. Rather, in preferred embodiments, merely a comparison of contents of a few important registers and/or memory contents is provided. The important registers in question may, for example, be the program counters and/or the stack pointers of the virtual machines. An example of important memory contents that are compared with one another in some embodiments of the invention is the most recent ("uppermost") element in the stacks of the virtual machines at the time in question.

[0013] Since the virtual machines on the portable data carrier are executed by a single processor, an interleaved program sequence generally takes place. That does not, however, exclude individual operations being executed truly in parallel if the processor of the portable data carrier is equipped to do so.

[0014] Checking of the operating states of the virtual machines may be carried out at any of the times when the virtual machines should have identical states if operating correctly. Although checking is not in principle tied, therefore, to the instruction boundaries of the program executed, in preferred embodiments it is provided that this checking is performed after each execution of an instruction of the program by the virtual machines. In alternative embodiments, the comparison of the virtual machines either may be carried out as soon as parts of instructions have been executed or may not be carried out until several instructions have been executed in each case.

[0015] Preferably, each instruction of the program is executed first by the first virtual machine and then by the second virtual machine. In some embodiments, execution of the instruction by the first virtual machine is first completed before the processor of the portable data carrier begins to execute the instruction using the second virtual machine. In other embodiments, on the other hand, the processor may execute each of a number of

portions of the instruction first on the first virtual machine and then on the second virtual machine, provided, however, that the first virtual machine does not lag behind the second virtual machine.

[0016] Owing to the use of at least two – and in some embodiments more – virtual machines, the computing capacity available for each virtual machine is correspondingly reduced. With regard to the actual program run time, however, it should be borne in mind that, in a typical portable data carrier, a great deal of time is required for write operations to a non-volatile memory of the data carrier.

[0017] In preferred embodiments, therefore, it is provided that the virtual machines access a common heap in the non-volatile memory of the portable data carrier, with write operations being performed by only one of the virtual machines. The other virtual machine(s) may either skip the write operation entirely or, instead of performing the write operation, may check whether the location that is to be written to in the memory already contains the value that is to be written. If the program to be executed contains a large number of write operations to the heap, these will require a considerable proportion of the total run time. Through the use of the embodiment of the invention just described, that portion of the total run time remains unchanged whereas only the purely computing time of the processor – which, as mentioned, matters less – increases.

[0018] The portable data carrier according to the invention is preferably in the form of a chip card or a chip module. The computer program product according to the invention has program instructions for implementing the method according to the invention. Such a computer program product may be a physical medium, for example a semiconductor memory or a diskette or a CD-ROM, on which a program for executing a

method according to the invention is stored. The computer program product may, however, alternatively be a non-physical medium, for example a signal transmitted via a computer network. The computer program product may especially be intended for use in connection with the production and/or initialisation and/or personalisation of chip cards or other data carriers.

[0019] In preferred embodiments, the data carrier and/or the computer program product have features corresponding to the features described above and/or to the features mentioned in the dependent method claims.

[0020] Further features, advantages and objects of the invention will be apparent from the following detailed description of an illustrative embodiment and a number of alternative embodiments. Reference will be made to the schematic drawings, in which:

[0021] Figure 1 is a block diagram showing functional units of a portable data carrier according to an illustrative embodiment of the invention,

[0022] Figure 2 is a conceptual illustration of components that are active when the program is being executed by the portable data carrier,

[0023] Figure 3 is a flow diagram of a main loop which is followed for each program instruction during execution of the program,

[0024] Figure 4 is a flow diagram of the checking of the operating states of the virtual machines for correspondence.

[0025] In the illustrative embodiment under consideration, the data carrier 10 shown in Figure 1 is in the form of a chip card conforming to the *Java Card* standard. The data carrier has, on a single semiconductor chip, a processor 12, a plurality of memory areas implemented in various technologies, and an interface circuit 14 for contactless or

contact-bound communication. In the illustrative embodiment under consideration, a working memory 16, a read-only memory 18 and a non-volatile memory 20 are provided as memory areas. The working memory 16 is in the form of RAM, the read-only memory 18 in the form of mask-programmed ROM and the non-volatile memory 20 in the form of electrically erasable and programmable EEPROM. Write accesses to the non-volatile memory 20 are relatively time-consuming and require, for example, thirty times as long as a read access.

[0026] In the read-only memory 18 – and partly also in the non-volatile memory 20 – there is an operating system 22 which provides a multitude of functions and services. The operating system 22 comprises *inter alia* a code module 24 that implements a virtual machine – a JCVM (*Java Card Virtual Machine*) in the illustrative embodiment under consideration.

[0027] Figure 1 shows by way of example a program 26 to be executed which is located in the non-volatile memory 20 and which, in the illustrative embodiment under consideration, is in the form of a *Java Card Applet.* The program 26 may also be held partly or completely in the read-only memory 18, and further programs for execution by the portable data carrier 10 may be provided. One area in the non-volatile memory 20 is reserved as a heap 28 in order for objects and other data structures to be held during execution of the program.

[0028] In order to execute the program 26 under the control of the operating system 22 the processor 12 starts two instances of the code module 24 each of which forms a virtual machine VM, VM'. As shown in Figure 2, the two virtual machines VM, VM' execute one and the same program 26 which is present in the non-volatile memory 20

only once. When the two virtual machines VM, VM' fetch an instruction of the program 26, therefore, they access identical addresses in the non-volatile memory 20.

[0029] As the program runs, the two virtual machines VM, VM' perform access operations to the common heap 28. Once again, the objects and data structures stored in the heap 28 are each present only once. The first virtual machine VM performs both read operations R and write operations W on the heap 28. The second virtual machine VM', on the other hand, although performing read operations R', does not perform any write operations but, rather, performs verification operations V.

[0030] The virtual machines VM, VM' each have their own registers, Figure 2 showing for each of the latter a program counter PC, PC' and a stack pointer SP, SP'. Those registers are disposed in the working memory 16 or are implemented by registers of the processor 12. Each virtual machine VM, VM' further has its own stack ST, ST' each disposed in a respective area of the working memory 16. The most recent ("uppermost") entries in the stacks ST, ST' at the time, to which the respective stack pointers SP, SP' point, are labelled @SP and @SP' in Figure 2.

[0031] As a modification of the illustration shown in Figure 1, the virtual machines VM, VM' may be disposed in separate hardware: in separate memories 20 which may also be assigned to separate processors. It may also be provided that the virtual machines VM, VM' be in the form of hardware components.

[0032] When the program 26 is being executed, the operating system 22 follows the loop shown in Figure 3. One pass of the loop is made for each instruction of the program 26. The instruction is first executed in step 30 by the first virtual machine VM. There are no differences here compared with the execution of a program in a prior art system by

a single virtual machine. In particular, the first virtual machine maintains its registers PC and SP and the stack ST and, where appropriate, performs a read operation R from and/or a write operation W to the heap 28.

[0033] When execution of the instruction by the first virtual machine VM has been completed, in step 32 the second virtual machine VM' executes the same instruction of the program 26 again. In this case also, maintenance of the registers PC' and SP', maintenance of the stack ST' and, where appropriate, an operation R' of reading from the heap 28 are performed in the usual manner.

[0034] Execution of the instruction by the second virtual machine VM' differs, however, from execution of the instruction by the first virtual machine VM in that, instead of any write operation which may be specified by the instruction, a comparison operation V is performed in which the value that is actually to be written to the heap 28 is compared with the current contents of the heap 28 at the address that is to be written to. If the calculation operations of the two virtual machines VM, VM' correspond, then the first virtual machine VM has already written the value that is now determined in step 32 by the second virtual machine VM' to the heap 28 in step 30. The verification operation V in step 32 therefore yields a correspondence, and the calculation sequence is continued. If, on the other hand, a difference is found between the values in step 32, that indicates a malfunction of one of the virtual machines VM, VM'. Execution of the program is then aborted as being defective. That possibility is indicated in Figure 3 by a dashed-line arrow.

[0035] After the program instruction has been executed by both virtual machines VM, VM', in step 34 it is examined whether the operating states reached by the two

virtual machines correspond. For that purpose, in the illustrative embodiment described herein only some of the register and memory values are examined for correspondence, as shown in Figure 4. First, in sub-step 34.1, it is examined whether the two program counters PC, PC' have the same value after the instruction has been executed. If that is the case, in sub-step 34.2 examination of the two stack pointers SP, SP' for correspondence takes place. If that test also is successful, in sub-step 34.3 it is examined whether the most recent entries @SP, @SP' in the stacks ST, ST' at the time, that is to say, the entries to which the stack pointers SP, SP' point, are identical.

[0036] If a correspondence has been found in all three queries 34.1, 34.2, 34.3, step 34 (Figure 3) assumes correct execution of the program by the two virtual machines VM, VM'. A return is then made to the start of the loop, and the next instruction of the program 26 is executed first by the first and then by the second virtual machine VM, VM'.

[0037] If, however, a difference is found in one of the three sub-steps 34.1, 34.2, 34.3 during checking of the operating states, that indicates a malfunction of one of the two virtual machines VM, VM'. That in turn is regarded as an indication of a fault or of an attack on the hardware of the portable data carrier 10. Since the processor 12 performs steps 30 to 32 in strict succession, when an attack occurs, for example by a flash of light, only the operation of one of the two virtual machines VM, VM' is affected. Even in the event of a rapid succession of light flashes it would be improbable that both virtual machines VM, VM' would be disrupted in the same manner.

[0038] If a difference in the operating states is found in step 34, execution of the program is aborted as being defective. The operating system 22 then puts the data carrier

10 into a secure state. It is to be particularly noted in this connection that, after a program abort, the data carrier 10 is not intended to perform any more output operations. Depending on the security requirements to be met by the data carrier 10, it may be provided that the data carrier 10 is ready for use again after a normal reset, or a special enable procedure may be required, or the data carrier 10 may be completely deactivated.